Revista Cubana de Ingeniería Vol. IX, No. 3, septiembre - diciembre, 2018, pp. 20 - 32

Artículo Original

AES T-Caja protegido contra ataques de fallo simple

AES T-Box protected against single fault attacks

Ernesto Gil Aranguren[1], Carlos M. Legón Perez[1], Humberto Diaz Pando[1]
[1]Universidad Tecnológica de la Habana, José Antonio Echeverría, Cujae, La Habana, Cuba
Correo electrónico: egil@ceis.cujae.edu.cu

Recibido: 25 de febrero de 2018 Aprobado: 11 de julio de 2018

Resumen
La criptografía es una de las herramientas para garantizar los atributos de seguridad tales como: confidencialidad, integridad, disponibilidad y no repudio de la información. El AES es uno de los algoritmos de cifrado simétrico en bloque más utilizados en la actualidad y se pueden encontrar numerosas soluciones y dispositivos criptográficos. Los ataques de canal colateral en especifico el DFA, así como los métodos de inducción de fallos en dispositivos hardware cada vez son más frecuentes por lo que aplicar contramedidas a esta situación se hace indispensable para mantener los requerimientos de seguridad. Este trabajo tiene como objetivo aplicar una contramedida contra fallo simple, los que constituyen los modelos de fallo más frecuentes a una implementación de AES utilizando la variante T-Box desarrollada para MicroBlaze. La contramedida se modificó para poder adaptarla a la variante de AES del componente y se logró mantener la efectividad de la contramedida; además se evidenció un incremento en cuanto al uso de memoria en un 13 % y a un aumento del tiempo de ejecución en una relación de 3 veces la velocidad inicial del componente.

Palabras claves**:** AES, DFA, fallo simple, contramedidas, MicroBlaze

Abstract
Cryptography is one of the tools to guarantee security attributes such as: confidentiality, integrity, availability and non-repudiation of information. The AES is one of the most widely used block symmetric encryption algorithms used today and numerous cryptographic solutions and devices can be found. Collateral channel attacks in specific DFA as well as the methods of inducing failures in hardware devices are becoming more frequent so applying countermeasures against this situation becomes indispensable to maintain the security requirements. This work aims to apply a countermeasure against simple failure, which are the most frequent failure models to an implementation of AES using the T-Box variant developed for MicroBlaze. The countermeasure was modified to be able to adapt it to the AES variant of the component and it was possible to maintain the effectiveness of the countermeasure, in addition it was evidenced an increase in the use of memory in 13 % and an increase of the execution time in a relation of 3 times the initial speed of the component.

Key words**:** AES, DFA, simple fault, countermeasure, MicroBlazents

INTRODUCCIÓN

El desarrollo de las telecomunicaciones ha propiciado un incremento exponencial de la información que se intercambia a diario. Para garantizar que dicha información se transmita segura por los distintos canales se emplea la criptografía. Mediante el uso de la criptografía se pueden garantizar los atributos de seguridad: confidencialidad, integridad, disponibilidad, y no repudio [1]. Dentro de la criptografía, la criptografía simétrica se utiliza para garantizar la confidencialidad de la información mientras se traslada por canales inseguros. Uno de los algoritmos más utilizados es AES (Estándar de Cifrado Avanzado, por sus siglas en ingles), un algoritmo simétrico que utiliza 128, 192, 256 bits de tamaño de clave [2].

Este algoritmo se ha implementado en diferentes tecnologías de hardware y en distintos entornos de software. Los autores del algoritmo *rijndael* [3] base del algoritmo AES, propusieron para plataformas de 32 bits la variante de implementación T-caja. Esta variante tiene como principal característica, mejorar los tiempos de respuesta disminuyendo los cálculos complejos del algoritmo mediante tablas de acceso rápido (look-up table). T-caja se ha logrado implementar tanto en software [4] como hardware específicamente FPGA donde se destaca el trabajo de [5], además, se pueden encontrar trabajos de hardware que implementan T-caja en [6], [7], donde se obtuvieron excelentes resultados de velocidad de procesamientos. En [8] se desarrolló un componente de cifrado simétrico AES utilizando la implementación T-caja.

En los últimos años se han reportado una serie de trabajos sobre un nuevo tipo de ataque a los algoritmos criptográficos, estos se denominan ataques de canal colateral [9]. Estos ataques difieren de los clásicos en que no se basan en debilidades matemáticas del algoritmo criptográfico si no que aprovechan las fugas físicas provocadas por las características de la tecnología donde se despliegan los servicios criptográficos que contienen información sobre el parámetro secreto (en general la clave). Entre los tipos de ataques, clasificados por fugas, se encuentran DFA (fallos durante la ejecución que generan resultados erróneos), DPA (consumo de potencia), Timing Attack (tiempo que demora la ejecución de ciertas operaciones criptográficas), EMA (emanación electromagnética), entre otros [10]–[12].

Existen numerosos reportes de ataques DFA al AES, que explican los métodos para obtener la clave del algoritmo AES a partir de fallas inducidas en el algoritmo en lugares específicos, en particular, se destacan los trabajos de Dusart y Giraud [13], [14] . Por otra parte, se pueden encontrar en otros trabajos resultados sobre los métodos de inducción de fallas en diferentes dispositivos mediante el uso de un cañón láser y un osciloscopio entre otros mecanismos [15]–[17]. Ante estos ataques DFA al AES, se han desarrollado contramedidas para contrarrestar estas vulnerabilidades. En [18] se plantea una contramedida contra fallo simple, la cual consiste en chequear los diferenciales de entrada-salida en distintos puntos del algoritmo que permitan reconocer si ha ocurrido algún fallo durante la ejecución de este. Esta contramedida posee una fácil implementación y no requiere de un consumo mayor de almacenamiento, lo que no significa una pérdida sustancial en la velocidad de procesamiento. Sin embargo, esta contramedida está pensada para la implementación estándar de AES y no para la implementación T-caja.

Este trabajo se propone como objetivo aplicar la contramedida propuesta por [18] a la implementación de AES T-caja de [8] para evaluar su aplicabilidad y contrastar su desempeño en plataformas de 32 bits; obteniéndose resultados satisfactorios al aplicar la contramedida y mantener los niveles de protección para fallo simple, así como aumentar en cierto modo el tiempo de ejecución y el consumo de memoria. El trabajo se estructura de ahora en adelante de la siguiente manera: Descripción del AES T-caja y sus implementaciones, DFA al AES, Contramedidas al DFA, Solución propuesta, Resultados, Conclusiones.

MATERIALES Y MÉTODOS

Descripción del AES T-caja y sus implementaciones

El Estándar de Cifrado Avanzado (Advanced Encryption Standard, AES por sus siglas en inglés), es un criptosistema de bloque, con un esquema de clave simétrica, es decir, que utiliza la misma clave para cifrar y descifrar [2]. El AES procesa bloques de datos de 128 bits y utiliza claves de tamaño 128, 192 y 256 bits. Utiliza como datos de entrada un mensaje o texto plano, y una clave secreta para el proceso de cifrado, de manera que utiliza esta misma clave para devolver el texto plano en el proceso de descifrado. Ambos elementos son modificados por cuatro transformaciones fundamentales: *SubByte, ShiftRow, MixColumns*, y *AddRoundKey*, en el caso del proceso de cifrado. El proceso de descifrado es muy similar, pero las transformaciones son ligeramente diferentes, denominándose operaciones inversas: *InvSubByte, InvShiftRow, InvMixColumns* y *InvAddRoundKey* [2].

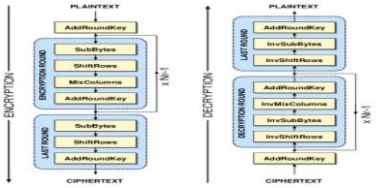

Fig. 1. Diagrama de funcionamiento del AES

En la figura 1 se observa el diagrama de bloques que muestra el funcionamiento del algoritmo AES. Como se puede apreciar las operaciones de cifrado poseen una operación inversa para el descifrado, lo cual puede incurrir en un mayor número de operaciones, en general en el momento de implementar dicho algoritmo. El AES es una modificación que se realizó del algoritmo de cifrado simétrico Rijndael diseñado por [3]. Los autores de este algoritmo documentaron en [3] una serie de consideraciones para su implementación en los distintos entornos de aplicación, o sea, implementación en software y hardware. Estas consideraciones han sido la base de algunas implementaciones actuales de este algoritmo. Dentro de estas se encuentra la implementación basada en T-Caja, la cual según los autores, ofrece mejor prestación en plataformas de 32 bit dado que combina en un simple acceso a memoria las operaciones de SubBytes y MixColumns. Esta última es la operación de mayor complejidad en el algoritmo.

Básicamente la modificación consiste en reducir las operaciones de SubBytes y MixColumns en una look-up table eliminando así la multiplicación en GF(2^8) de la operación MixColumns. En la ecuación 1 se muestra el proceso original de transformación de una columna cualquiera, y en la ecuación 2 se observa entonces cómo se descompone la ecuación 1 para cada columna de la matriz de MixColumns.

$$\begin{bmatrix} e_{15}^i \\ e_{14}^i \\ e_{13}^i \\ e_{12}^i \end{bmatrix} = \begin{bmatrix} '02' & '03' & '01' & '01' \\ '01' & '02' & '03' & '01' \\ '01' & '01' & '02' & '03' \\ '03' & '01' & '01' & '02' \end{bmatrix} \otimes \begin{bmatrix} SB(d_{15}^i) \\ SB(d_{10}^i) \\ SB(d_5^i) \\ SB(d_0^i) \end{bmatrix} \tag{1}$$

Por último, en la ecuación 3 como resultado de la descomposición realizada en 2 se observa cómo se conforma el resultado para una columna dada de la matriz State que originalmente se expuso en la ecuación 1 utilizado la look-up table.

$$\begin{bmatrix} '02' \\ '01' \\ '01' \\ '03' \end{bmatrix} \otimes \left[SB(d_{15}^i)\right] \oplus \begin{bmatrix} '03' \\ '02' \\ '01' \\ '01' \end{bmatrix} \otimes \left[SB(d_{10}^i)\right] \oplus \begin{bmatrix} '01' \\ '03' \\ '02' \\ '01' \end{bmatrix} \otimes \left[SB(d_5^i)\right] \oplus \begin{bmatrix} '01' \\ '01' \\ '03' \\ '02' \end{bmatrix} \otimes \left[SB(d_0^i)\right] \tag{2}$$

$$T_0(a) = \begin{bmatrix} '02' \bullet SB(a) \\ SB(a) \\ SB(a) \\ '03' \bullet SB(a) \end{bmatrix} \quad \mathrm{T}_1(a) = \begin{bmatrix} '03' \bullet SB(a) \\ '02' \bullet SB(a) \\ SB(a) \\ SB(a) \end{bmatrix} \quad \mathrm{T}_2(a) = \begin{bmatrix} SB(a) \\ '03' \bullet SB(a) \\ '02' \bullet SB(a) \\ SB(a) \end{bmatrix} \quad \mathrm{T}_3(a) = \begin{bmatrix} SB(a) \\ SB(a) \\ '03' \bullet SB(a) \\ '02' \bullet SB(a) \end{bmatrix} \tag{3}$$

Esta implementación tiene la particularidad que requiere de memoria concretamente 4KB para las 4 tablas de cifrado y 4KB para las descifrado. Por tanto, no es posible implementarla en todos los dispositivos. Existe una variante propuesta también por [3] que reduce el consumo de memoria dado que solo utiliza una tabla en vez de

4, esta modificación reduce considerablemente el consumo de memoria a 1KB (2KB en total) y mantiene la reducción del tiempo de ejecución dado que solo agrega un sencillo mecanismo de rotación.

Es posible encontrar implementaciones basadas en T-caja, tanto en hardware como en software. En el caso de hardware en la tabla 1 se hace referencia a las implementaciones en FPGA de AES T-caja con la correspondiente velocidad de procesamiento y consumo de slices. En el caso de [8] como es un diseño hardware/software existe una latencia en la comunicación entre MicroBlaze y el modulo IP que encapsula la T-caja, además hay que considerar en el consumo de slices la cantidad de slices que ocupa MicroBlaze el cual es de 1863 slices.

En el caso de las implementaciones completamente en software, en [4] se realiza una mejora en cuanto al tiempo de ejecución del AES para distintas plataformas software. Entre estas se encuentra una implementación de AES T-caja que alcanza una velocidad de procesamiento de 16.8 Gbps en un GPU Nvidia 8800 GTX con frecuencia de 1,35 GHz. Ver tabla 1.

Tabla 1. Relación de implementaciones de AES T-caja en FPGA

Implementación	Dispositivo	Velocidad de proc.	Consumo de slices	Tipo de implementación
Rouvroy et al. [5]	Xilinx Spartan-3 XC3S50	208 Mbps	163	Hardware
Rouvroy et al. [5]	Xilinx Virtex-II XC2V40	358Mbps	146	Hardware
Pramstaller et al. [6]	XCV1000E	215 Mbps	1125	Hardware
Chodowiec &Gaj [7]	XC2S30-6	166Mbps	222	Hardware
Aranguren et al. [8]	Xilinx Spartan-3E XC3S500E	877Kbps	2049(187)	Hardware/Software

DFA AL AES

Como se ha mencionado anteriormente el uso de implementaciones de AES en dispositivos criptográficos propicia un escenario apropiado para la aparición de ataques de canal colateral y en particular el canal colateral de DFA. El AES es susceptible al DFA por su diseño y es posible encontrar una serie de trabajos desde el año 2003 que hacen referencia a esta vulnerabilidad. En los últimos años se han encontrado mejoras en los modelos de ataques y nuevos métodos de inducción de fallos lo cual hace que se incremente la atención a la seguridad contra este tipo de ataque.

Algunos de los trabajos DFA están basados en modelos teóricos de [19]–[22] mientras que otros han logrado realizar ataques efectivos sobre dispositivos ASIC y FPGA, utilizando los modelos teóricos de [2,8,28,46,49].

La clave del DFA consiste en 3 etapas:

1. Ejecutar el algoritmo criptográficos y obtener texto cifrado sin fallos.
2. Inducir fallos en una implementación criptográfica despreciando las condiciones de su ejecución y obtener texto cifrado con fallos utilizado las mismas entradas que en la anterior etapa.
3. Analizar la relación entre los textos cifrados con fallos y sin fallos para reducir el espacio de búsqueda de la clave [23].

Aunque la naturaleza de los fallos varía se pueden catalogar de la siguiente manera:

1. Fallo a nivel de bit: Este modelo de fallo asume la existen de un fallo en un bit determinado. Concretar este fallo es muy complejo dada la exactitud que se necesita del mecanismo de inducción que muchas ocasiones se modifica más de un bit [23].

2. Fallo a nivel de un byte: Es el modelo más práctico y más frecuentemente encontrado. Este modelo asume la modificación aleatoria de un byte cuyo valor de modificación es desconocido, lo cual deriva en otros submodelos y técnicas de inducción muy prácticas [23].

3. Fallo en múltiples byte: Este modelo asume que el fallo se propaga a más de un byte. Cada día se hacen más prácticos estos modelos, en buena medida porque trabajan con menos control de inducción de fallo [23].

En [19] se documenta un nuevo tipo de modelo de ataque en el cual el fallo es introducido en el proceso de expansión de la clave. Este tipo de ataque tiene la particularidad de que un solo fallo inducido en una subclave especifica puede generar todo el texto cifrado con fallo, además, dado que las subclaves se encuentran almacenadas en memoria, se puede incluso inducir el fallo posterior al proceso de generación de las subclaves y obtener el mismo resultado. Otra de las modificaciones al proceso de DFA al AES se puede encontrar en [20], este modelo parte de un solo texto cifrado defectuoso, y mediante algunas consideraciones permite obtener la clave en un espacio de tiempo práctico, esto reduce considerablemente la cantidad de elementos conocidos por el atacante para llevar a cabo la efectividad del ataque, y refuerza la necesidad de proteger las implementaciones de los dispositivos embebidos. Por otra parte en [21] se documenta un nuevo modelo de ataque donde se inserta un valor aleatorio de fallo en la ronda 8, luego en dos fases y utilizando métodos probabilísticos se deduce los valores de la clave para el AES-128. Otra de las modificaciones al ataque de diferencial de fallas se introduce en [22], donde el comportamiento de la falla puede ser analizado mediante técnicas DPA: el promedio de ruido y la prueba de hipótesis por correlación. Este trabajo presenta el análisis de intensidad de fallas diferenciales, el cual combina los principios del DPA y la inducción de fallas, teniendo en cuenta que las fallas muchas veces son sesgadas. Por último, en [24] se realizó un estudio para demostrar la factibilidad de los ataques DFA en FPGA y la seria amenaza que este implica.

Pero los avances no son solo en materia de modelos de ataques DFA, también existen una serie de trabajos enfocados a inducir el fallo. En [25] se describe un mecanismo para inducir fallos en la implementación de AES tanto a nivel de software como hardware, este último diseñado en FPGA mediante emanaciones electromagnéticas. Sin embargo, el fallo para el caso de las FPGA puede llegar incluso de una forma más directa y es el caso del trabajo reseñado en [26], el cual induce el fallo durante el proceso de reconfiguración del FPGA vía Ethernet, no obstante, puede ser usado este esquema si el bitstream es modificado en el origen.

Particularmente entre los reportes de ataques de DFA al AES se pueden mencionar [27], dado que el mismo documenta el ataque de [20] pero en la implementación T-Caja para FPGA. Este trabajo aplica el ataque mencionado satisfactoriamente sobre arquitecturas bien conocidas de las FPGA de Xilinx mediante la modificación de su biststream. Se demuestra que el atacante puede encontrar toda la clave para sus distintos tamaños una vez que el atacante pueda modificar las T-Caja almacenadas.

Contramedidas al DFA

Como se puede apreciar los ataques de DFA son amenazas ya concretas y en especial al AES y sus aplicaciones en sus distintas plataformas tanto en hardware como software, dada la naturaleza de los fallos los cuales pueden ser temporales o permanente [23]. La mayoría de las contramedidas se diseñan para detectar los fallos temporales, los cuales son los más peligrosos, pues no dejan evidencias [23]. Las contramedidas se clasifican según [23] en dos tipos fundamentales: CED (Detección de Error Concurrente, por sus siglas en ingles). Dentro de los CED se encuentran las categorías de

- Redundancia de hardware: Se caracteriza por duplicar el módulo de la operación para evaluar si hay diferencias entre las salidas de los módulos de operación. Por lo general este tipo de contramedida produce un incremento del doble en cuanto al diseño original del AES y logra detectar hasta el 90 % de los fallos. Teniendo en cuenta la capacidad del atacante de inducir fallos, el mismo puede inducir el fallo en más de un módulo para evitar este tipo de contramedida [23].
- Redundancia de tiempo: Se caracteriza por evaluar dos veces la misma operación con el objetivo de comparar las salidas respectivas [23].
- Redundancia de información: Este tipo de contramedida se basa en EDS (Código de Detección de Errores, por sus siglas en ingles). Se basa en el uso de unos pocos bits generados a partir del mensaje de entrada, luego estos bits se propagan junto con el mensaje hasta el final del algoritmo donde se validan con el resultado final obtenido. Para esta contramedida se proponen los códigos de paridad y los códigos robustos [23].

La contramedida propuesta por [18] tiene contemplada la verificación mediante la relación diferencial de entrada y salida en cada una de las operaciones del AES y en el proceso de expansión de la clave, de esta manera queda cubierto cada uno de los momentos donde el atacante puede inducir un fallo. Esta contramedida se clasifica como de redundancia de información, aunque para que su implementación no requiera de un incremento de diseño original del AES, sobre todo en dispositivos, se utiliza solo un byte para almacenar el diferencial de entrada y salida. En el trabajo de [24] se aclara que la contramedida propuesta detecta el 100 % de los fallos simples.

Solución propuesta

Para proteger la implementación T-caja de AES documentada en [8], con la contramedida de [18] es necesario realizar modificaciones a dicha implementación. Esta contramedida se basa en insertar operaciones sencillas dentro del AES, sin embargo, es necesario analizar si se puede implementar sobre la implementación de [8]. Esto implica modificar ligeramente las operaciones de chequeo propuestas por la contramedida y que aumente el espacio de memoria requerido, así como el tiempo de ejecución del algoritmo con respecto a la implementación de referencia.

La contramedida debe ser aplicada en cuatro niveles: protección de la expansión de la clave, protección a nivel de operación (SubBytes, AddRoundKey), protección a nivel de ronda y protección a nivel de algoritmo. Se excluyen las operaciones de MixColumns y ShitfRows debido a que, según los autores de la contramedida, en estas operaciones los diferenciales de entrada-salida son 0 en ambos casos. Estos casos descritos previamente son para el diseño clásico del algoritmo. Para la variante con una T-caja se necesita hacer modificaciones a la contramedida mínimo a nivel de operación.

Protección en la expansión de la clave: Como la implementación seleccionada no realiza ninguna modificación en la operación de KeyExpansion, esta se implementa tal cual está documentada en [18].

Protección de la transformación *SubBytes-MixColumns:* La transformación SubByte opera sobre los valores de la matriz de estado intermedio, sustituyendo byte a byte sus valores, por sus correspondientes salidas en la S-Box. Esto implica que, si se logra inyectar un fallo durante la función de sustitución, ya sea el índice de entrada a la S-Box o el valor de la S-Box accedido (salida). Dado que esta operación es no lineal para poder chequear si ha ocurrido este tipo de fallo se debe agregar una matriz de chequeo, que almacene la operación $\oplus$ de cada uno de los posibles 256 valores de entrada a la S-Box, con sus correspondientes salidas, debido a que en la implementación seleccionada están mezcladas SubBytes y MixColumns en una T-caja, la construcción de la matriz de chequeo (T-caja-C) se realiza análogamente como lo propone la contramedida pero con la T-caja. En la tabla 2 se refleja un ejemplo de la construcción sencilla de esta matriz de chequeo.

Tabla 2. Matriz de chequeo de la transformación SubByte-MixColumns

Entrada (T-caja)	Salida (T-caja)	Salida (T-caja-C)
0x00(0)	0xC66363A5	0xC66363A5
0xFF(255)	0x2c16163aU	0x2C1616C5

Las condiciones que se deben cumplir para comprobar si hay fallo o no en la transformación, se expresan en las siguientes ecuaciones:

$$E_{T\text{-}caja}=E_0 \oplus E_1 \oplus \dots E_{15} \quad (4)$$

$$S_{T\text{-}caja}=S_0 \oplus S_1 \oplus \dots S_{15} \quad (5)$$

$$D_{T\text{-}caja}= T\text{-}caja\text{-}C(d_{15}) \oplus T\text{-}caja\text{-}C(d_{14}) \oplus \dots T\text{-}caja\text{-}C(d_0) \quad (6)$$

$$D_{T\text{-}caja\text{-}C}=E_{SB} \oplus S_{SB} \quad (7)$$

Donde $E_{T\text{-}caja}$ y $S_{T\text{-}caja}$ representan los valores diferenciales de entrada y salida respectivamente T-caja. *T-caja-C(d_i)* representa la salida de la tabla de chequeo para cada valor de la matriz State previo a su sustitución

por T-caja. $D_{T\text{-}caja}$ es el valor diferencial de los valores de chequeo correspondientes para cada byte de la matriz de estado en la ronda. Si la ecuación 7 se cumple, entonces no hubo alteraciones en la sustitución por la T-caja, de manera que antes de continuar el cifrado se debe chequear esta condición. Esto se debe a que el diferencial de MixColumns es 0 según los autores de la contramedida [18].

Protección de transformación AddRoundKey: En este caso tampoco la variante compacta hace modificaciones en esta operación, por tanto, la comprobación por parte de la contramedida se mantiene sin cambios. Esta comprobación se realiza mediante la ecuación 10 y como se observa el diferencial de entrada-salida debe ser igual a la subclave de ronda.

$$I_{ARK}=I_0 \oplus I_1 \oplus ... I_{15} \tag{8}$$

$$O_{ARK}=O_0 \oplus O_1 \oplus ... O_{15}= I_0 \oplus RK_0 \oplus I_1 \oplus RK_1 \oplus ... \ I_{15} \oplus RK_{15} \tag{9}$$

$$D_{ARK}=I_{ARK} \oplus O_{ARK}=RK_0 \oplus RK_1 \oplus ... RK_{15} \tag{10}$$

Protección a nivel de ronda: La protección a nivel de ronda está encaminada a detectar un fallo que ocurra durante la ejecución de la ronda. Esto implica que si el fallo se produce antes de comenzar la ronda, la contramedida no detectará este fallo en el nivel de ronda. Si el fallo ocurriese al terminar la ronda tampoco y sería el caso de que se inyectara en la ronda siguiente.

Cada ronda intermedia del algoritmo, es decir, excepto la primera y la última, depende de los resultados de las operaciones SubBytes y AddRoundKey. Significa que, si se desea chequear el comportamiento de una ronda, el diferencial entre la entrada y la salida de cada una debe ser igual al valor diferencial entre la entrada y la salida de la sustitución mediante la T-caja calculado previamente al chequeo de ronda, y el valor diferencial de la clave de ronda. En las siguientes ecuaciones se muestra el proceso de chequeo de la ronda:

$$D_{RND}=D_{T\text{-}caja} \oplus D_{ARK} \tag{11}$$

Donde D_{RND} representa la suma $\oplus$ entre los valores I_{RND} entrada de la ronda y O_{RND} salida de la ronda. $D_{T\text{-}caja}$, D_{ARK} son los valores diferenciales de entrada y salida de la sustitución en T-caja y la operación AddRoundKey respectivamente.

Al no tener la última ronda la operación MixColumns es necesario extraer el valor original de SubBytes contenido en la operación MixColumns, esta problemática ha sido resuelta en la variante compacta. El resultado de un valor de la tabla de acceso rápido SB-MC no es más que el resultado de SubBytes multiplicado la primera columna de MixColumns, por tanto, si se quisiera obtener el valor real de SubBytes, basta con escoger del valor de retorno el byte que haya sido multiplicado por 1. Por ejemplo, para el valor de 0x00(0) el resultado de la sustitución por SB-MC seria 0xC66363A5 y de este se extrae el 2do byte, el cual coincide con el valor real de SubBytes (0x63) de 0x00.

Protección a nivel de algoritmo: Esta debe detectar todas las alteraciones encontradas por las contramedidas anteriores y aquellas que ocurran entre rondas, que no estén al alcance de las anteriores. Como se analizó anteriormente, la ronda inicial depende del texto plano y la clave; cada ronda intermedia depende de las operaciones de sustitución mediante T-caja y AddRoundKey, y la última depende de SubBytes y AddRoundKey. Para chequear el cifrado a nivel de algoritmo es necesario obtener el diferencial de entrada y salida del algoritmo, es decir, el resultado de la suma $\oplus$ entre el texto plano y el texto cifrado. Este valor diferencial se compara con el resultado diferencial entre todas las rondas del algoritmo según se expresa en la siguiente ecuación:

$$D_{ALG}=I_{ALG} \oplus O_{ALG} \tag{12}$$

A continuación, se expresa el valor diferencial de las rondas:

$$D_X=D_{RND}^{0} \oplus D_{RND}^{i} \oplus D_{RND}^{10} \quad (13)$$

La condición de chequeo del algoritmo es la siguiente:

$$D_X=D_{ALG} \quad (14)$$

Si se cumple la ecuación, entonces no se produjeron problemas durante el cifrado, de lo contrario la contramedida detectará los errores.

RESULTADOS

Para comprobar la eficacia de la contramedida aplicada a la solución propuesta fue necesario realizar un conjunto de experimentos induciendo fallos en los distintos niveles de protección. Los niveles de protección son: nivel de operación, nivel de ronda y nivel de algoritmo. Además, comprobar si al aplicar las fallas descritas en los modelos comentados previamente la contramedida detecta dichas fallas.

Para medir efectividad no basta con inducir fallos de manera aleatoria y particular dependiendo del modelo de fallo, sino que es necesario cuantificar para los distintos esquemas de fallo su nivel de cobertura. Para esto existe la métrica cobertura de fallo (FC, por sus siglas en inglés) [probably], cuyo valor se calcula como se aprecia en la ecuación 15.

$$FC = 1 - \frac{Tnodetectados}{Ttotal - Tcorrectos} \quad (15)$$

Donde Tnodetectados son aquellos fallos que la contramedida no logra detectar y provocan una salida fallida, Ttotal son los fallos totales inducidos y Tcorrectos, los fallos que no provocan salida fallida [probably].

En cuanto a los tipos de fallos inducidos se utilizaron dos variantes: fallo simple (modificación de un solo byte) y fallos múltiples, variante (DM0) que se puede encontrar en [23] debido a que la contramedida seleccionada, según sus autores, propone cobertura de fallo del 100 % y se desea analizar el comportamiento de la misma para una variante de fallos múltiples.

Existen distintos esquemas de modelos de fallos, estos se definen por la zona del proceso de cifrado que afectan. En este trabajo se utiliza los 3 esquemas siguientes:

- Aleatorio: simula un fallo simple de hardware no intencional que provoca salidas fallidas del algoritmo.
- Esquema 1: modifica un byte de la matriz estado justo antes del MixColumns de la 8va. ronda.
- Esquema 2: modifica un byte de la matriz estado justo antes del MixColumns de la 9na. ronda.

En cuanto a la posición del fallo dentro de la matriz estado:

- Para el caso del esquema aleatorio, se modificarán los 16 bytes.
- Para el esquema 1 se modificarán las posiciones (0,0), (1,0), (2,0), (3,0) en diferentes momentos.
- Para el esquema 1 se modificarán las posiciones (0,0), (1,0), (2,0), (3,0) en diferentes momentos.

Con los resultados de la tabla 3, se confirma que la aplicación de la contramedida en la variante AES compacto. cumple satisfactoriamente su objetivo. Además, que la contramedida captura 100 % de los fallos simples inducidos en la implementación.

Tabla 3. Resultados de inducción de fallos en la solución propuesta

Esquema	Valor del fallo	Posición	Tipo de fallo	Cobertura de fallo (%)
Aleatorio	0...255	-	Fallo simple	100
Esquema 1	0...255	(0,0),(0,1),(0,2),(0,3)	Fallo simple	100
Esquema 2	0...255	(0,0),(0,1),(0,2),(0,3)	Fallo simple	100

Los fallos inducidos para el esquema aleatorio fueron ubicados en las distintas posiciones de la T-caja, posiciones utilizadas dentro del proceso de cifrado. En el caso del AddRoundKey, los fallos inducidos fueron ubicados en cada una de las posiciones de la subclave de ronda. Por otra parte, los fallos a nivel de la ronda se ubicaron indistintamente en las operaciones de SB-MC y AddRounKey.

Los esquemas 1 y 2 no tienen aplicación tal cual se propone para la implementación de AES T-caja, por tanto, los experimentos se realizaron mediante la modificación de un byte de la matriz estado justo antes de ejecutar la operación de sustitución en la tabla T. En este sentido los resultados obtenidos fueron los mismos que los del esquema aleatorio.

Con el objetivo de evaluar el desempeño de la solución propuesta para analizar el coste, tanto en tiempo de ejecución como en consumo de memoria, se utilizó la plataforma de pruebas de [8] en su variante totalmente software. Dicha plataforma consta del procesador de propósito general de tipo RISC MicroBlaze con 50MHZ de frecuencia de reloj y memoria RAM de 32KB. Este procesador se incluye como parte de la plataforma base de Xilinx para sus FPGA para este trabajo se utilizó la Spartan-3E.

El AES T-caja propuesta por [3] e implementado en FPGA por [5] tiene como ventaja la reducción del tiempo de ejecución del algoritmo dado que evita la complejidad de la operación MixColumns al almacenarla en las tablas de acceso rápido. La modificación hecha en [8] propuesta por [3] reduce el uso de memoria dado que utiliza 1 y no 4 T-caja aunque agrega un mecanismo de permutación de 4 bytes, lo cual disminuye ligeramente el tiempo de ejecución del AES con T-caja tradicional. En la propuesta de este trabajo, sin embargo, se han agregado las operaciones propias de la contramedida implementada y es necesario evaluar en cuanto a tiempo de ejecución las diferencias entre la variante de AES con una T-caja dado que esta sí puede ser ejecutada sin dificultad en MicroBlaze.

En la tabla 4 se presenta una comparación en ciclos de reloj que muestra las diferencias en tiempo de ejecución de las variantes de AES analizadas. Se utilizarán ciclos de reloj para tener valores más descriptivos dado que para un solo bloque el tiempo que consume es muy pequeño. Como se puede apreciar en los resultados de la tabla 4, el AES con una T-caja es 3 veces más rápido que el AES con una T-caja protegido, sin embargo, ambos AES con T-caja son mucho más rápidos que la implementación estándar. Si se analiza el tiempo que tarda en cifrar un bloque serian 761,07e-5s, 6.68e-05s y 21,326e-05s para la implementación estándar, el AES con una T-caja de una tabla y AES con una T-caja protegido respectivamente. Para tener una idea más clara de velocidad de procesamiento de ambas variantes se debe hacer una estimación de la escalabilidad de estos resultados para un mayor número de bloques, utilizando la ecuación 15.

$$TMP_{CIF}=CANT_{BLK}*(CICLOS_{RELOJ}/FREC_{PROC}) \quad (15)$$

Donde TMP_{CIF} se refiere al tiempo de cifrado total para una cantidad específica de bloques a cifrar, $CANT_{BLK}$ representa la cantidad de bloques, los ciclos de reloj que demora cada variante se representa en $CICLOS_{RELOJ}$ y por último, la $FREC_{PROC}$ se refiere al valor asociado que representa la frecuencia del procesador MicroBlaze que en este caso es un valor constante 50MHz(50,000,000).

En la tabla 5 se muestra la escalabilidad del tiempo de ejecución de ambas variantes teniendo en cuenta la ecuación 15 para distintas cantidades de bloques a cifrar.

Tabla 4. Comparación en ciclos de reloj entre AES Estándar, AES compacto de una tabla y solución propuesta

Ciclos de reloj/ Variante de AES	AES Estándar [2]	AES T-caja [8]	Solución propuesta
	380535	3340	10663

Tabla 5. Tiempos de cifrado para distintas cantidades de bloques de las implementaciones de AES analizadas

Variante analizadas	Cantidad de bytes cifrados		Velocidad
	1000(16KB)	1000000(16MB)	
AES Estándar [2]	7,6107s	7610,7s (2h 6mins)	2,053030KB/s
AES T-caja [8]	0,0668s	66,8s (1min 6,8s)	233,90625KB/s
Solución propuesta	0,21326s	213,26s (3min 33s)	73,265625KB/s

En el caso de analizar la velocidad de procesamiento de ambas variantes se tiene los siguientes valores: 2,053030KB/s (16,42424Kb/s) para la implementación estándar de AES, 233,90625KB/s (1871,25Kb/s) para el AES de una T-caja y 73,265625KB/s (586,125Kb/s) para el AES de una T-caja protegido. Se puede apreciar claramente que en todos los casos el factor alrededor de 3 está presente en todos los resultados de comparación entre ambas con T-caja, ya sea por velocidad de procesamiento y tiempo de ejecución para uno o más bloques. Además, las variantes con T-caja son mucho mejores que la implementación estándar para este entorno de hardware.

Otro factor importante de la compactación para el desarrollo de sistemas embebidos es el consumo del recurso de memoria. Para este escenario el consumo de memoria RAM también es un tema a analizar, pues muchas veces las implementaciones no se encuentran aisladas en estos dispositivos sino que es necesario tener lógica del negocio implementado, además de otros algoritmos criptográficos. Para este análisis se utiliza nuevamente la implementación estándar, el AES con una T-caja y AES con una T-caja protegido.

Al observar los valores de la tabla 6, se aprecia cómo el consumo de memoria de la implementación estándar y con una T-caja protegido están por encima de 20KB, sin embargo, en el caso de la variante protegido posee una velocidad de ejecución superior a la velocidad de ejecución de la implementación estándar. Además, no hay tantas diferencias entre los valores del variante con una T-caja y la con una T-caja protegida, dado que la última es modificación de la primera, por lo que se puede apreciar que la aplicación de la contramedida solo repercutió en un incremento de 2,33KB.

Tabla 6. Comparación en consumo de memoria entre AES Estándar, AES con una T-caja y solución propuesta

Consumo de memoria RAM (Bytes)/Variante de AES	AES Estándar [2]	AES T-caja [8]	Solución propuesta
	21408	18960	21352

La estrategia detrás de la implementación T-caja es la de reducir el tiempo de ejecución mediante el almacenamiento de los cálculos correspondientes. Esto genera un incremento de la memoria en función de una mejora en los tiempos de respuesta. En tal sentido, de manera general, cuando se requiere incrementar la velocidad de procesamiento sin cambiar el algoritmo de una operación en específico, hay un aumento del consumo de memoria. En la ecuación 16 se cuantifica este fenómeno mediante la variable I_R la cual expresa si el consumo de memoria se justifica con un incremento de la velocidad en la ejecución.

$$I_R=CONS_{MEM}*TMP_{EXE} \quad (16)$$

Donde I_R es el valor índice, $CONS_{MEM}$ representa el consumo de memoria y TMP_{EXE} se refiere al tiempo de ejecución para cifrar un bloque.

Los resultados de la tabla 7 permiten corroborar que la diferencias en tiempo de ejecución y consumo de memoria RAM entre las variantes compactas, son sustancialmente mejor que la implementación estándar en este entorno de hardware, al observarse una caída brusca en el índice de memoria/tiempo para las variantes compactas con respecto a la estándar.

En resumen, la variante compacta protegida tuvo un incremento de alrededor 3 veces el tiempo de ejecución con respecto a la variante compacta de una tabla y un incremento del 12,61 % del consumo de memoria compara con variante de una tabla. Sin embargo, al analizar estos resultados con la implementación estándar se observa que la reducción del tiempo de ejecución es del 97,2 % y el consumo de memoria es casi el mismo.

Tabla 7. Comparación en índice de rendimiento memoria/tiempo entre AES Estándar, AES con una T-caja de una tabla y AES con una T-caja protegido

(Área/tiempo)/ Variante de AES	AES Estándar [2]	AES T-caja [8]	Solución propuesta
	162,93	1,27	4,55

CONCLUSIONES

A partir de los resultados de este trabajo se arriban a las siguientes conclusiones:

- Es posible aplicar la contramedida de diferencial de entrada-salida en la variante de AES con T-caja.
- Se obtuvo el 100 % de acierto al encontrar los fallos inducidos en las distintas posiciones.
- La variante de AES con una T-caja protegida es alrededor de 3,19 veces más lento que la variante con una T-caja y consume un 7,48 % más de memoria en el entorno de prueba de este trabajo.
- Ambas variantes con T-caja son sustancialmente mejor en su desempeño que la implementación estándar de AES para el entorno de prueba de este trabajo, ya que en las dos variantes la reducción del tiempo de ejecución está por encima del 97 % y el consumo de memoria no fue excesivamente mayor.

- La variante de amortiguación del 5 % muestra mayores registros de aceleración que el 2 % presentando, lo cual implica mayores zonas en el talud de fallas probables con asentamientos diferenciales locales.

RECOMENDACIONES

Para continuar el estudio del tema de este trabajo se proponen las siguientes recomendaciones:

- Utilizar la técnica de codiseño para reducir el tiempo de ejecución de la variante propuesta en este trabajo.
- Evaluar la contramedida para fallos múltiples e identificar cuáles fallos múltiples logra detectar.
- Agregar a la propuesta en la medida de lo posible otra contramedida que proteja ante otros tipos de ataques de canal colateral.

REFERENCIAS

1. Menezes J, Van Oorschot PC, Vanstone SA. Handbook of applied cryptography. CRC press, 1996.
2. Fips P.197, Adv. Encryption Stand. AES. 2001, 26 pp.
3. Daemen J, Rijmen V. The design of Rijndael. Information security and cryptography. Springer Berlin. 2002.
4. Osvik DA, Bos JW, Stefan D, Canright D. Fast software AES encryption, in International Workshop on Fast Software Encryption. 2010, pp. 75–93.
5. Rouvroy G, Standaert FX, Quisquater JJ, Legat JD. Compact and efficient encryption/decryption module for FPGA implementation of the AES Rijndael very well suited for small embedded applications, in Information Technology: Coding and Computing, Proceedings. ITCC 2004. International Conference on. 2004, 2:583–587.
6. Pramstaller N, Wolkerstorfer J. A universal and efficient AES co-processor for field programmable logic arrays, Field Program. Log. Appl. 2004, pp. 565–574,
7. Chodowiec P, Gaj K. Very compact FPGA implementation of the AES algorithm, in International Workshop on Cryptographic Hardware and Embedded Systems. 2003, pp. 319–333.
8. Aranguren EG, García YM, et al. Componentes de cifrado simétrico sobre microblaze basados en los estándares AES y GOST, Rev. Telem Tica. 2014,13(1):46–62.
9. Standaert FX. Introduction to side-channel attacks, in Secure Integrated Circuits and Systems, Springer. 2010, pp. 27–42.
10. Martínez Bejarano R. Evaluación de la seguridad de sistemas embebidos ante ataques EMA. 2013.
11. Selmane N. Attaques en Fautes Globales Et Locales Sur Les Cryptoprocesseurs: Mise en Oeuvre Et Contremesures. 2010.
12. Zhou Y, Feng D. Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing. IACR Cryptol. EPrint Arch. 2005, p. 388.
13. Dusart P, Letourneux G. Vivolo O. Differential fault analysis on AES, in International Conference on Applied Cryptography and Network Security., 2003, pp. 293–306.
14. Giraud C. DFA on AES, in International Conference on Advanced Encryption Standard. 2004, pp. 27–41.
15. Tajik S, Lohrke H, Ganji F, Seifert JP, Boit C. Laser fault attack on physically unclonable functions, in 2015 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC). 2015, pp. 85–96.
16. Höller A, Krieg A, Rauter T, Iber J, Kreiner C. QEMU-based fault injection for a system-level analysis of software countermeasures against fault attacks, in Digital System Design (DSD), 2015 Euromicro Conference on, 2015, pp. 530–533.
17. Hayashi Y, Homma N, Mizuki T, Aoki T, Sone H. Fundamental study on fault occurrence mechanisms by intentional electromagnetic interference using impulses, in 2015 Asia-Pacific Symposium on Electromagnetic Compatibility (APEMC). 2015, pp. 585–588.
18. Park J, Bae K, Choi Y, Choi D, Ha J. A fault-resistant AES implementation using differential characteristic of input and output, J Internet Serv Inf Secur. 2012, 2(3):93–109.

19. Kim CH. Improved differential fault analysis on AES key schedule, IEEE Trans. Inf. Forensics Secur. 2012, 7(1): 41–50.
20. Fuhr T, Jaulmes E, Lomné V, Thillard A. Fault attacks on aes with faulty ciphertexts only, in Fault Diagnosis and Tolerance in Cryptography (FDTC), 2013 Workshop on. 2013, pp. 108–118.
21. Tunstall M, Mukhopadhyay D, Ali S. Differential fault analysis of the advanced encryption standard using a single fault, in IFIP International Workshop on Information Security Theory and Practices. 2011, pp. 224–233.
22. Ghalaty NF, Yuce B, Taha M. Schaumont P. Differential fault intensity analysis, in Fault Diagnosis and Tolerance in Cryptography (FDTC). 2014 Workshop on. 2014, pp. 49–58.
23. Ali S, Guo X, Karri R. Mukhopadhyay D. Fault attacks on AES and their countermeasures, in Secure System Design and Trustable Computing. Springer. 2016, pp. 163–208.
24. Momeni H. Masoumi M, Dehghan A. A practical fault induction attack against an FPGA implementation of AES cryptosystem, in Internet Security (WorldCIS), 2013 World Congress on. 2013, pp. 134–138.
25. Dehbaoui A, Dutertre JM, Robisson B. Tria A. Electromagnetic transient faults injection on a hardware and a software implementations of AES, in Fault Diagnosis and Tolerance in Cryptography (FDTC), 2012 Workshop on. 2012, pp. 7–15.
26. Johnson AP, Saha S, Chakraborty RS, Mukhopadhyay D, Gören S. Fault attack on AES via hardware Trojan insertion by dynamic partial reconfiguration of FPGA over ethernet, in Proceedings of the 9th Workshop on Embedded Systems Security. 2014, p. 1.
27. Aldaya AC, Sarmiento AJC, Sánchez-Solano S. AES T-Box tampering attack, J. Cryptogr. Eng. 2016, 6(1):31–48